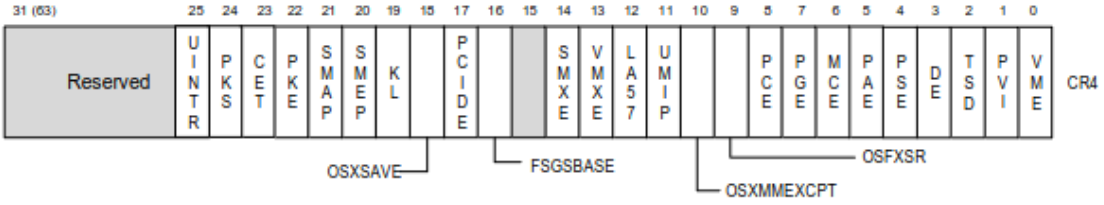
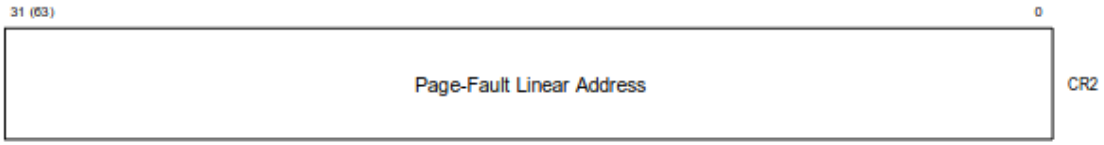
Registros que contienen la ubicación de la estructura de datos utilizada en segmentación : TR, GDTR, IDT, LDTR.

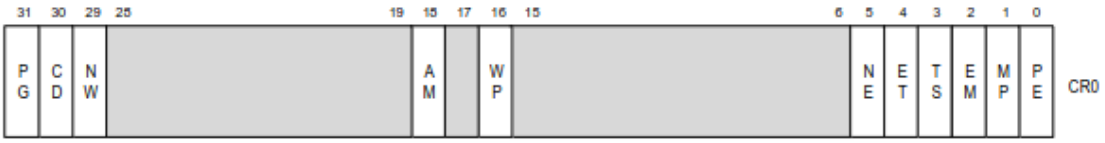
Instrucciones LGDT, LIDT, LTR : load estructura correspondiente
* 32 bits para la base address y 16 bits para el límite de cada estructura

Registros de control: Determinan las características de la tarea en ejecución. Tienen 32 bits TODAS.

CR0	CR1	CR2	CR3	CR4
Flags que controlan el modo de operación(protected , pagination, etc)	Reservado	Guarda la dirección virtual que causó un page fault	Contiene la dirección física de la estructura de paginación. 20 bits + significativos(parte alta) = base address 12 bits – significativos = se asumen como 0	Contiene flags que habilitan distintas extensiones de la arquitectura







Cap 3 – Manejo de memoria Modo Protegido

Segmentación	Paginación
Isolás secciones de la memoria en data, codigo y pila → multiples programas corren en el mismo procesador y no se pisan entre sí	Implementa un modo de páginas on demand → Secciones(memoría virtual) de un programa en ejecución se mapean a la memoria física cuando se necesitan.
Uso obligatorio. No se puede deshabilitar.	Uso opcional.
La memoria = "Espacio de direcciones virtuales/lineales" se divide en secciones = segmentos Dirección lineal(far pointer) = segment_selector:offset con el segment selector vas al segmento, con el offset vas al byte que querés	Acá los segmentos se dividen en páginas de 4Kb que se guardan en la memoria física Cuando una tarea quiere acceder a una dirección virtual, el procesador la traduce a física y lee/escribe en memoria. Si no está mapeada → page fault pros: * read-write protection enforced on a page-by-page basis * two-level user-supervisor protection specified on a page-by-page basis

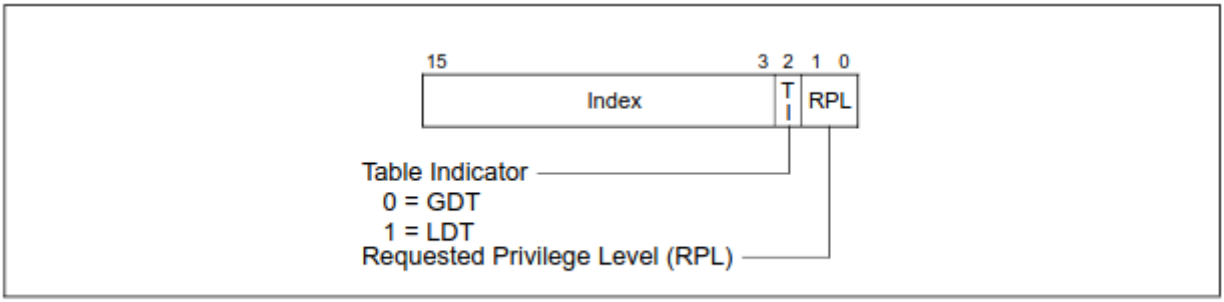


Figure 3-6. Segment Selector

Two kinds of load instructions for seg.sel
1. **Direct load instructions** such as the MOV, POP, LDS, LES, LSS, LGS, and LFS instructions. These instructions

explicitly reference the segment registers.
2. **Implied load instructions** such as the far pointer versions of the CALL, JMP, and RET instructions, the SYSENTER
and SYSEXIT instructions, and the IRET, INT n, INTO, INT3, and INT1 instructions. These instructions **change**
the **contents of the CS register** (and sometimes other segment registers) as an incidental part of their
operation

Visible Part	Hidden Part	
Segment Selector	Base Address, Limit, Access Information	CS
		SS
		DS
		ES
		FS
		GS

Figure 3-7. Segment Registers

1) Offset para buscar el segmento en la GDT

2) Chequea los permisos en el descriptor de segmento y ve si el offset está dentro del límite del segmento

3) Se suma la base descripta en el descriptor del segmento al offset y se consigue la dirección lineal

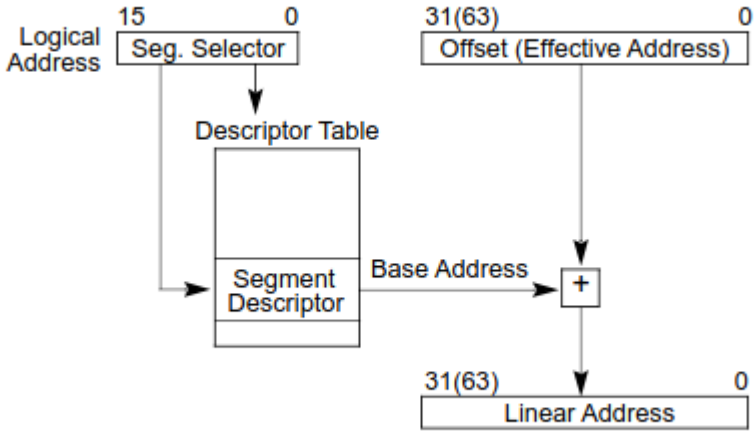
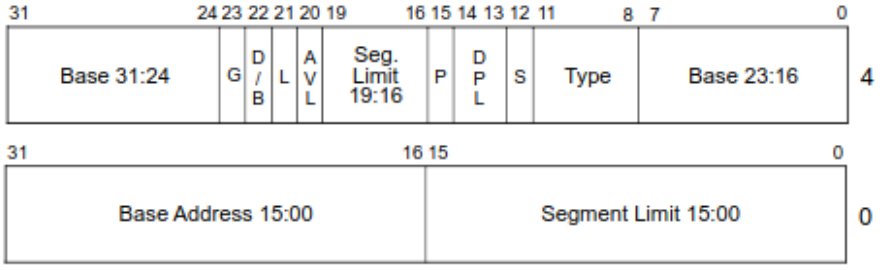


Figure 3-5. Logical Address to Linear Address Translation

Entrada en la GDT

GENERAL



L — 64-bit code segment (IA-32e mode only)
AVL — Available for use by system software
BASE — Segment base address
D/B — Default operation size (0 = 16-bit segment; 1 = 32-bit segment)
DPL — Descriptor privilege level
G — Granularity
LIMIT — Segment Limit
P — Segment present
S — Descriptor type (0 = system; 1 = code or data)
TYPE — Segment type

Figure 3-8. Segment Descriptor

Tipos de segmentos de **codigo y data (S = 1)**

Table 3-1. Code- and Data-Segment Types

Type Field					Descriptor Type	Description
Decimal	11	10 E	9 W	8 A		
0	0	0	0	0	Data	Read-Only
1	0	0	0	1	Data	Read-Only, accessed
2	0	0	1	0	Data	Read/Write
3	0	0	1	1	Data	Read/Write, accessed
4	0	1	0	0	Data	Read-Only, expand-down
5	0	1	0	1	Data	Read-Only, expand-down, accessed
6	0	1	1	0	Data	Read/Write, expand-down
7	0	1	1	1	Data	Read/Write, expand-down, accessed
		C	R	A		
8	1	0	0	0	Code	Execute-Only
9	1	0	0	1	Code	Execute-Only, accessed
10	1	0	1	0	Code	Execute/Read
11	1	0	1	1	Code	Execute/Read, accessed
12	1	1	0	0	Code	Execute-Only, conforming
13	1	1	0	1	Code	Execute-Only, conforming, accessed
14	1	1	1	0	Code	Execute/Read, conforming
15	1	1	1	1	Code	Execute/Read, conforming, accessed

Tipos de **segmento de sistema (S = 0)**

* Task-state segment (TSS) descriptor.
• Call-gate descriptor.
• Interrupt-gate descriptor.
* Trap gate descriptor
* Task gate descriptor

Table 3-2. System-Segment and Gate-Descriptor Types

Type Field					Description	
Decimal	11	10	9	8	32-Bit Mode	IA-32e Mode
0	0	0	0	0	Reserved	Reserved
1	0	0	0	1	16-bit TSS (Available)	Reserved
2	0	0	1	0	LDT	LDT
3	0	0	1	1	16-bit TSS (Busy)	Reserved
4	0	1	0	0	16-bit Call Gate	Reserved
5	0	1	0	1	Task Gate	Reserved
6	0	1	1	0	16-bit Interrupt Gate	Reserved
7	0	1	1	1	16-bit Trap Gate	Reserved
8	1	0	0	0	Reserved	Reserved
9	1	0	0	1	32-bit TSS (Available)	64-bit TSS (Available)
10	1	0	1	0	Reserved	Reserved
11	1	0	1	1	32-bit TSS (Busy)	64-bit TSS (Busy)
12	1	1	0	0	32-bit Call Gate	64-bit Call Gate
13	1	1	0	1	Reserved	Reserved
14	1	1	1	0	32-bit Interrupt Gate	64-bit Interrupt Gate
15	1	1	1	1	32-bit Trap Gate	64-bit Trap Gate

These descriptor types, a su vez, fall into two categories:

* System-segment descriptors point to system segments (LDT and TSS segments).
* Gate descriptors are in themselves“gates,” which hold pointers to procedure entry points in code segments (call, interrupt, and trap gates) or which hold segment selectors for TSS’s (task gates)

GDT – Global Descriptor Table

Todos los sistemas tienen que tener una

The base linear address and limit of the GDT must be loaded into the GDTR register

segment descriptors(entries en la GDT) are always 8 bytes long

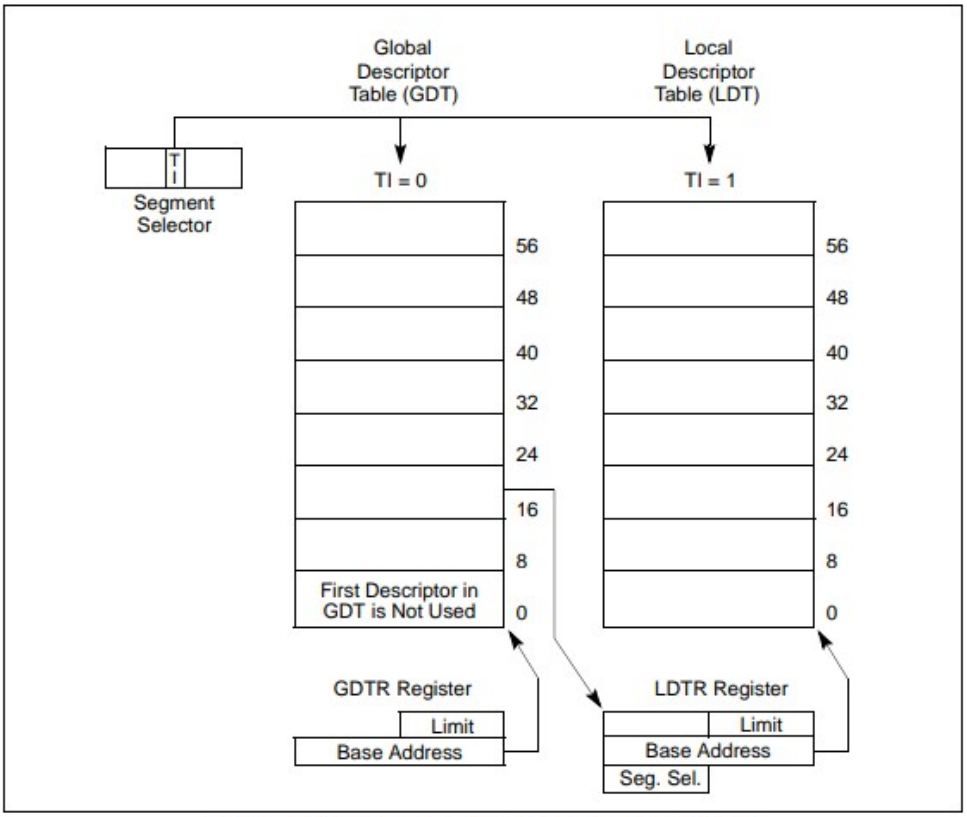


Figure 3-10. Global and Local Descriptor Tables

Cap 4 – Paginacion

Each paging-structure entry contains a physical address, which is either the address of another paging structure or the address of a page frame. In the first case, the entry is said to reference the other paging structure; in the latter, the entry is said to map a page.

With 32-bit paging, each paging structure comprises 1024 = 210 entries. For this reason, the translation process uses 10 bits at a time from a 32-bit linear address. Bits 31:22 identify the first paging-structure entry and bits 21:12 identify a second. The latter identifies the page frame. Bits 11:0 of the linear address are the page offset within the 4-KByte page frame.

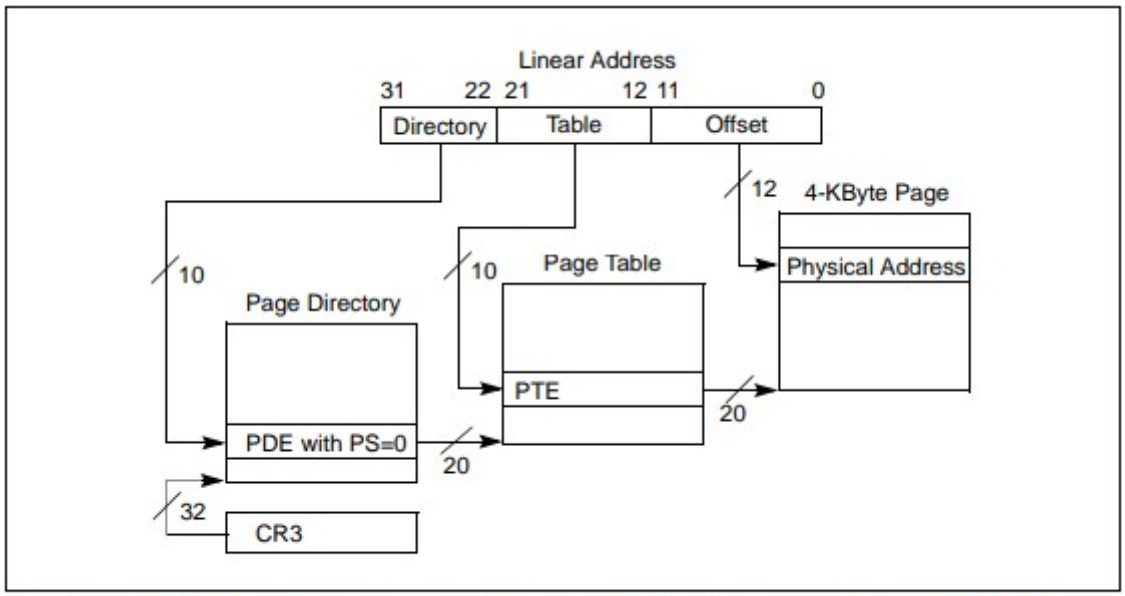


Figure 4-2. Linear-Address Translation to a 4-KByte Page using 32-Bit Paging

31–12	11–10	9	8	7	6	5	4	3	2	1	0	
Address of page directory[1]	Ignored						PCD	PWT	Ignored			CR3
Bits 31:22 of address of 4MB page frame; Reserved (must be 0); Bits 39:32 of address[2]	PAT	Ignored	G	1	D	A	PCD	PWT	U/S	R/W	1	PDE: 4MB page
Address of page table	Ignored			0	Ign	A	PCD	PWT	U/S	R/W	1	PDE: page table
Ignored											0	PDE: not present
Address of 4KB page frame	Ignored		G	PAT	D	A	PCD	PWT	U/S	R/W	1	PTE: 4KB page
Ignored											0	PTE: not present

Figure 4-4. Formats of CR3 and Paging-Structure Entries with 32-Bit Paging

Access rights
CPL < 3 => supervisor
CPL = 3 => user mode
Si las entradas PDE y PTE difieren en privilegios, se queda con el de menor privilegio
PF (page fault) entre otras> pagina no mapeada, no hay privilegios necesarios

Page-fault exceptions can be caused by 1) an access to a linear address that was not mapped 2) No hay los permisos necesarios para acceder a la pagina

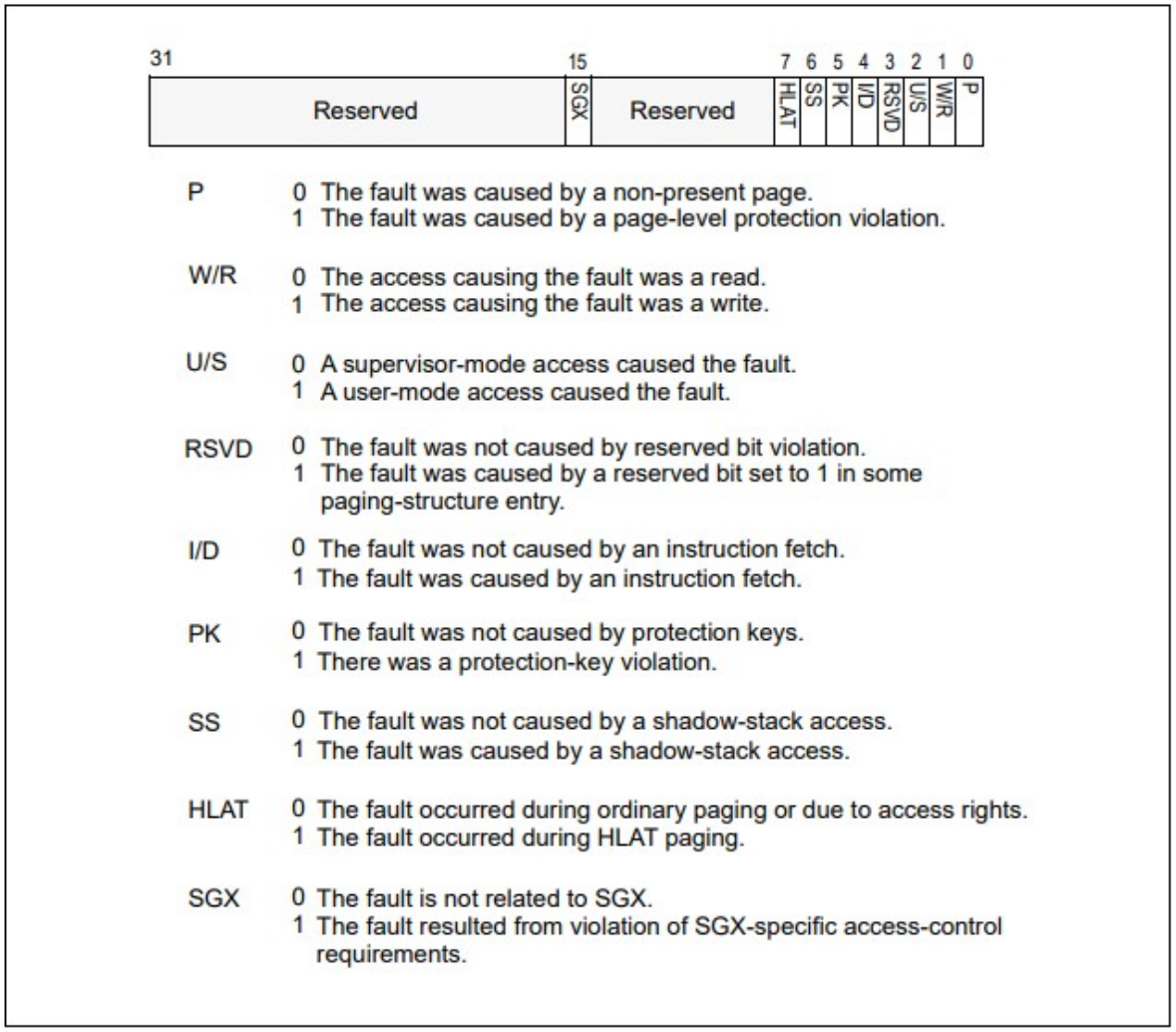


Figure 4-12. Page-Fault Error Code

Translation Lookaside Buffers (TLBs)

A processor may cache information about the translation of linear addresses in translation lookaside buffers (TLBs). In general, TLBs contain entries that map page numbers to page frames;

Se almacenan la direc lineal para la PDT, la PT y los bits de la direccion fisica 12 a 31 del descriptor de la pagina.

Como cada tarea tiene su propia estructura o sea su CR3, cada escritura en el registro CR3 flushea el contenido de la TLB

Capitulo 5 – Interrupciones

Hay 256 interrupciones disponibles. 0-19 ya definidas. De la 20 a 31 reservadas.
De 32 a 255 somos libres de usarlas

Table 6-1. Protected-Mode Exceptions and Interrupts

Vector	Mnemonic	Description	Type	Error Code	Source
0	#DE	Divide Error	Fault	No	DIV and IDIV instructions.
1	#DB	Debug Exception	Fault/ Trap	No	Instruction, data, and I/O breakpoints; single-step; and others.
2	—	NMI Interrupt	Interrupt	No	Nonmaskable external interrupt.
3	#BP	Breakpoint	Trap	No	INT3 instruction.
4	#OF	Overflow	Trap	No	INTO instruction.
5	#BR	BOUND Range Exceeded	Fault	No	BOUND instruction.
6	#UD	Invalid Opcode (Undefined Opcode)	Fault	No	UD instruction or reserved opcode.
7	#NM	Device Not Available (No Math Coprocessor)	Fault	No	Floating-point or WAIT/FWAIT instruction.
8	#DF	Double Fault	Abort	Yes (zero)	Any instruction that can generate an exception, an NMI, or an INTR.
9		Coprocessor Segment Overrun (reserved)	Fault	No	Floating-point instruction.[1]
10	#TS	Invalid TSS	Fault	Yes	Task switch or TSS access.
11	#NP	Segment Not Present	Fault	Yes	Loading segment registers or accessing system segments.
12	#SS	Stack-Segment Fault	Fault	Yes	Stack operations and SS register loads.
13	#GP	General Protection	Fault	Yes	Any memory reference and other protection checks.
14	#PF	Page Fault	Fault	Yes	Any memory reference.

Vector	Mnemonic	Description	Type	Error Code	Source
15	—	(Intel reserved. Do not use.)		No	
16	#MF	x87 FPU Floating-Point Error (Math Fault)	Fault	No	x87 FPU floating-point or WAIT/FWAIT instruction.
17	#AC	Alignment Check	Fault	Yes (Zero)	Any data reference in memory.[2]
18	#MC	Machine Check	Abort	No	Error codes (if any) and source are model dependent.[3]
19	#XM	SIMD Floating-Point Exception	Fault	No	SSE/SSE2/SSE3 floating-point instructions[4]
20	#VE	Virtualization Exception	Fault	No	EPT violations[5]
21	#CP	Control Protection Exception	Fault	Yes	RET, IRET, RSTORSSP, and SETSSBSY instructions can generate this exception. When CET indirect branch tracking is enabled, this exception can be generated due to a missing ENDBRANCH instruction at target of an indirect call or jump.
22-31	—	Intel reserved. Do not use.			
32-255	—	User Defined (Non-reserved) Interrupts	Interrupt		External interrupt or INT *n* instruction.

En la materia: Excepciones → Generadas por el procesador cuando se cumpla una condición
Ej. Bit de present P en 0
Interrupciones:
Interrupciones externas → PIC1, PIC2
Interrupciones internas → Generadas por software. Se llaman con INT n

Tipos de excepciones:

- Fault→ Excepción que puede corregirse para que el programa continúe su ejecución. El procesador guarda en la pila la direcci´on de la instrucci´on que produjo la falla.

- Traps → Excepci´on producida al terminar la ejecuci´on de una instrucci´on de trap.

- Aborts → Explota todo. Errores severos en el hardware.

Consideraciones para agregar interrupciones:

- Escribimos la rutina de atención de la interrupción. Se usa IRET ahora.
- Agregamos más entradas a la IDT con los descriptores correspondientes.
- Cargamos el descriptor de IDT en IDTR

Estado de la Pila

ANTES de atender la interrupción se PUSHEA
1) registro EFLAGS
2) registro CS
3) registro EIP
4) OP – dependiendo de las interrupciones, se pushea el error code. Ej el #PF pushea el error code.

El IRET saca todas estas cosas.

→ Las interrupciones se atienden a nivel de kernel ←

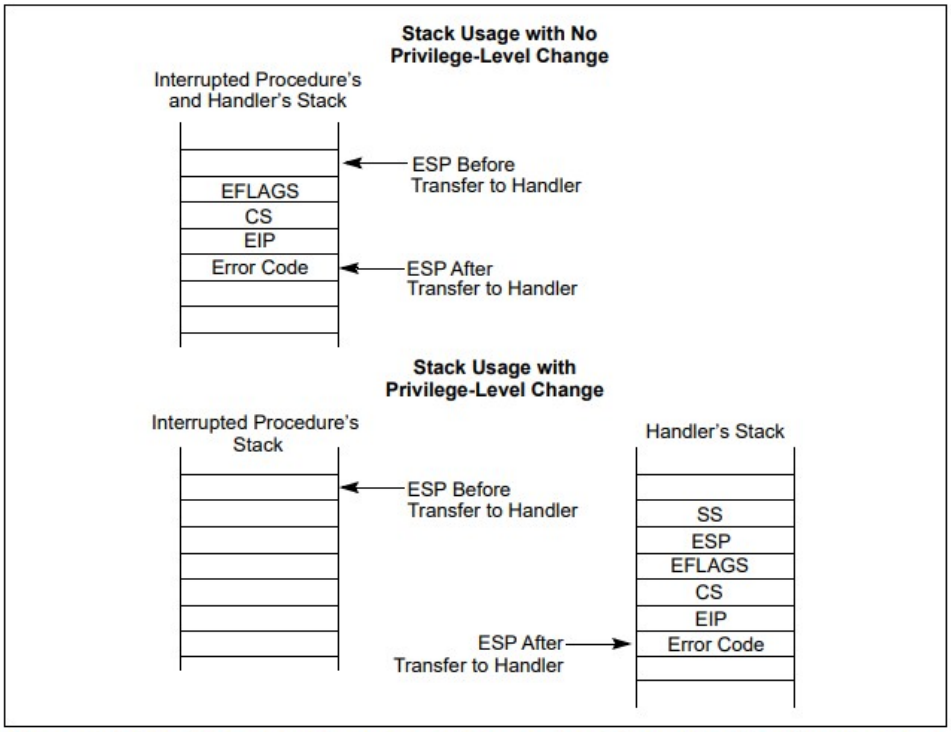


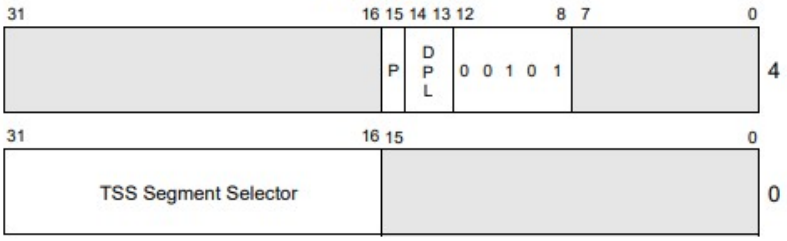
Figure 6-4. Stack Usage on Transfers to Interrupt and Exception-Handling Routines

Los descriptores de las interrupciones que vamos a usar en la materias

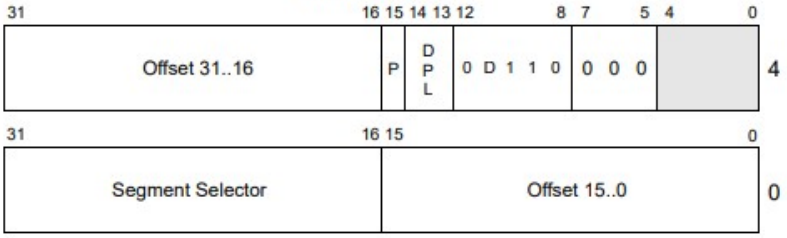
DPL	Descriptor Privilege Level
Offset	Offset to procedure entry point
P	Segment Present flag
Selector	Segment Selector for destination code segment
D	Size of gate: 1 = 32 bits; 0 = 16 bits

Reserved

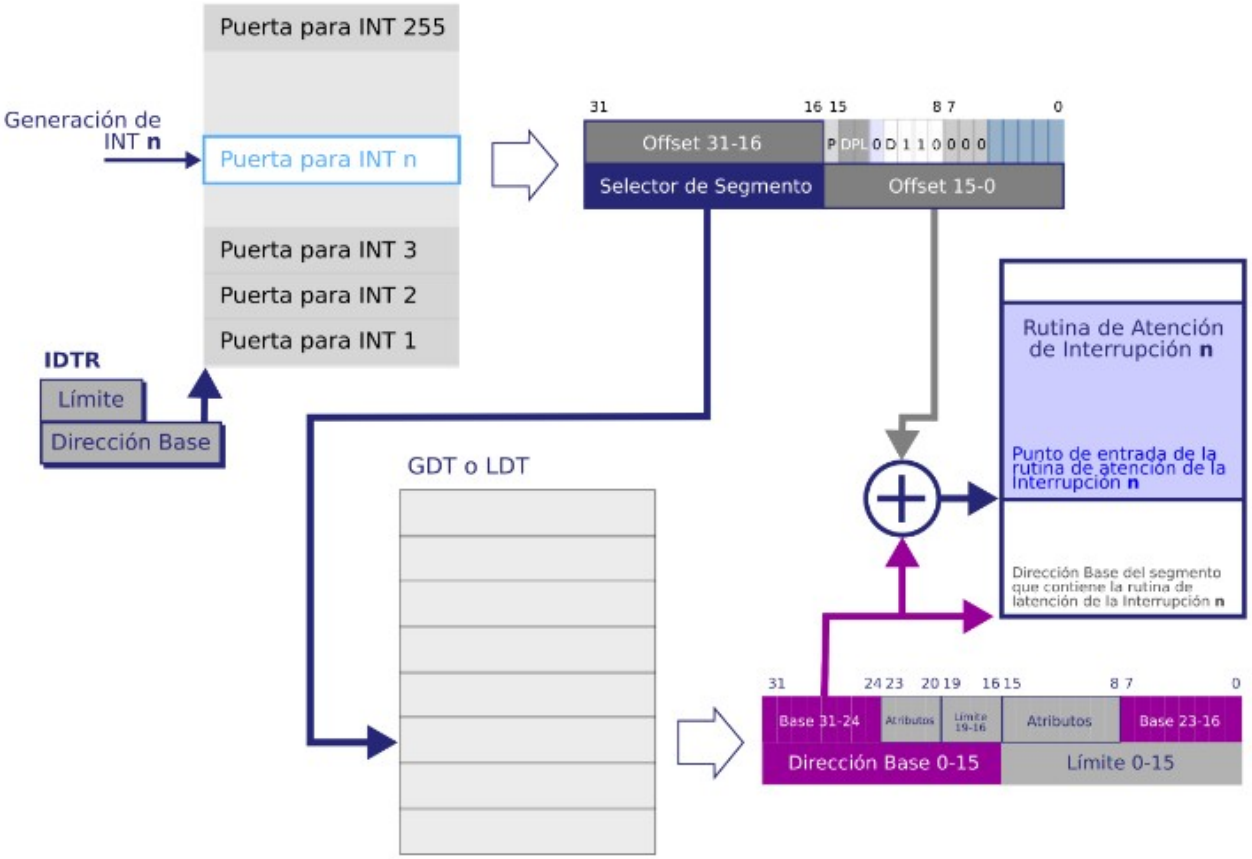
Task Gate



Interrupt Gate



En el taller usamos
Type 14 : 1110 (de los bits 8 a 12)
(haciendo referencia a la tabla 3-2)



Cap 8. Tareas

Tareas: Estructuras que te permiten ejecutar disintos programas “en simultaneo”.
Scheduler: Un modulo del software que decide qué tarea ejecutar

Las tareas tienen INSTANCIAS de un programa. O sea, corren el mismo codigo pero contienen datos distintos, se interrumpieron en distintos momentos, etc.

Las Tareas tienen

1) Espacio de Ejecución :
- Segmentos de codigo
- segmentos de datos/pila
- Mapa de paginas- CR3 con paginacion activa

2) Segmento de Estado(TSS)
*Almacena el estado de la tarea

* Registros de prop general y selectores de segmento
* Flags, CR3, EIP,LDTR y pilas de niveles de kernel.
Los registros de prop general de la tarea de nivel 3 quedan en la pila de nivel 0

1. Al iniciar las tareas:
   - completar **EIP**
   - completar **ESP** y **EBP**
   - completar selectores de segmento **CS**, **DS**, · · · ,**SS** (recordar RPL)
   - completar **CR3**
   - completar **EFLAGS** (*)

2. Al saltar por primera vez a una tarea:
   - tener un descriptor en la GDT de la tarea inicial
   - tener un descriptor en la GDT de la tarea a saltar
   - tener en **TR** algún valor válido (tarea inicial)

Cambios de contexto → En nuestra materia, se dan en las interrupciones de reloj

El TR tiene la info a la tarea actual.

Para inicializar una tarea:
- EIP

• ESP, EBP y ESP0 (puntero al tope de pila de nivel 0)

• Los selectores de segmento CS, DS, ES, FS, GS, SS, SS0 (selector de la pila de nivel 0)
• El CR3 que va tener la paginacion asociada a la tarea.
• EFLAGS en 0x00000202 para tener las interrupciones habilitadas

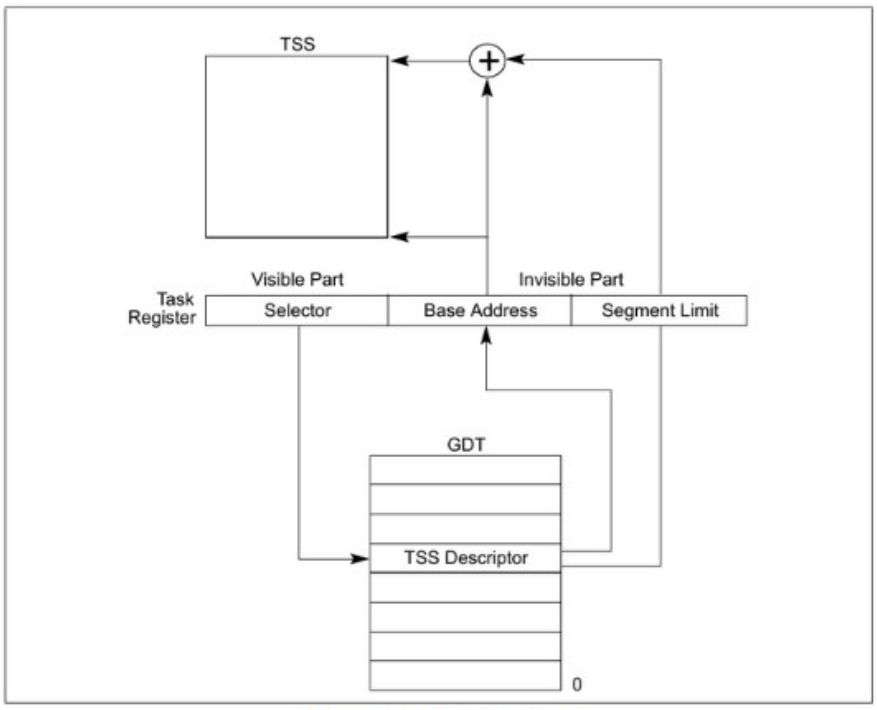


Figure 7-5. Task Register

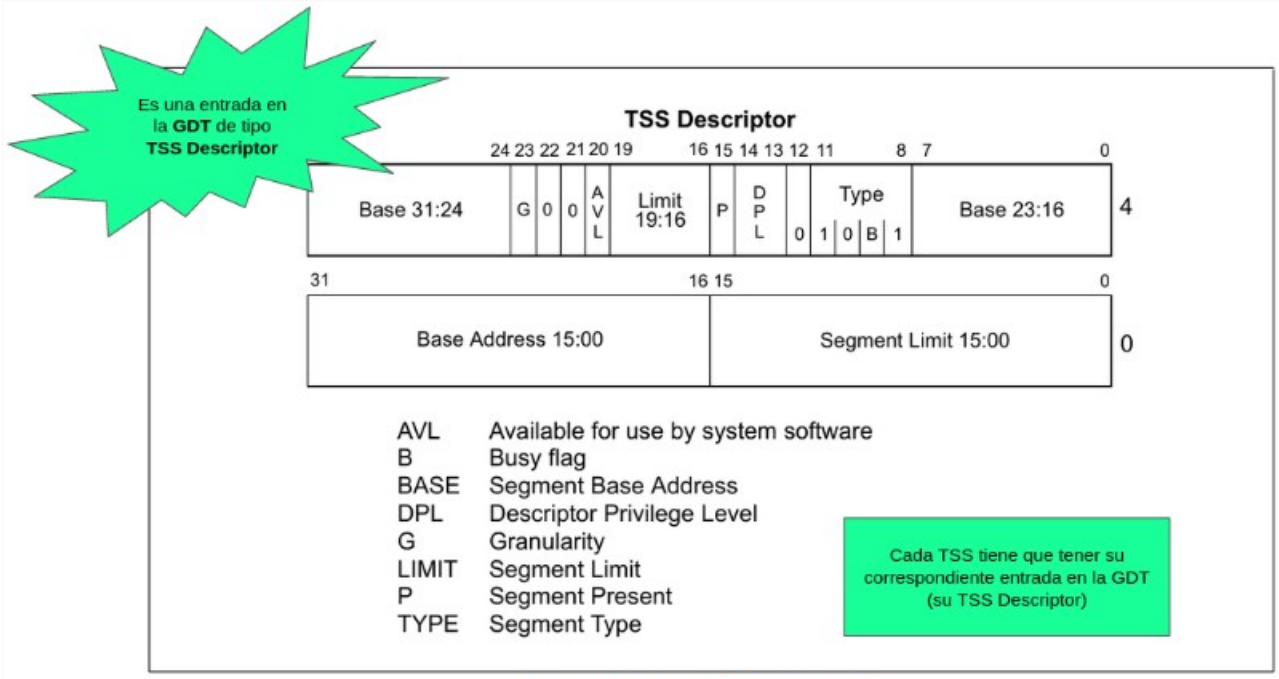


Figure 7-3. TSS Descriptor

Intercambio de Tareas

Para saltar a una tarea cualquiera es necesario poder modificar el selector de segmento.
- Usando: jmp <selector>:0. No es posible, ya que <selector> es fijo.
- Entonces, se debe usar memoria para indicar el selector de segmento para el intercambio.

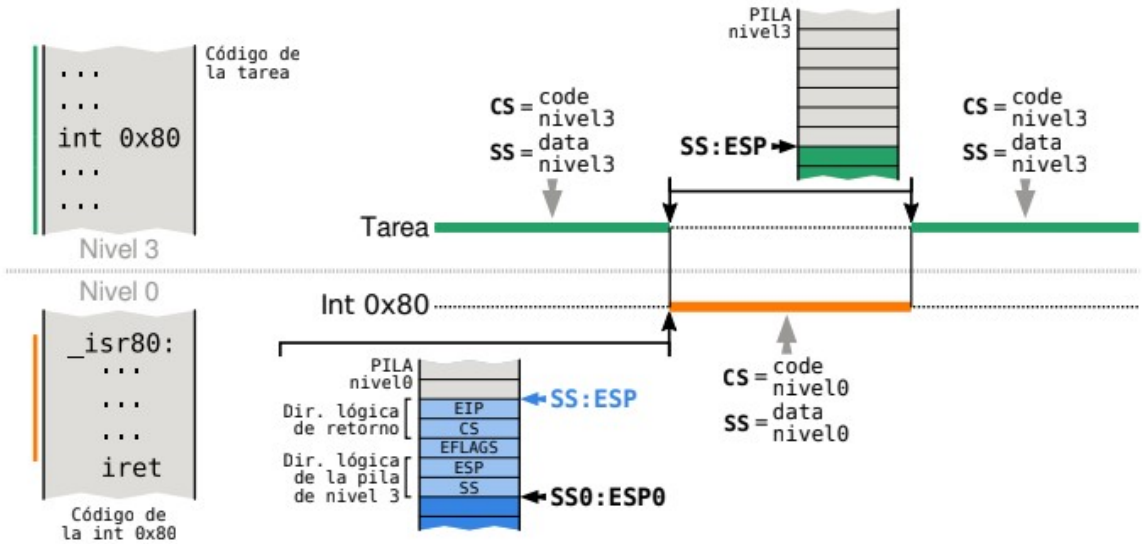
Se define en algun lugar del codigo la siguiente estructura:

```
offset: dd 0
selector: dw 0
```

La estructura definida se puede ver como una direccion logica de 48 bits en little endian
En la rutina se utiliza de la siguiente forma:

```
...
mov [selector], ax ; se carga el selector de segmento
jmp far [offset] ; se salta a la direccion logica definida
```

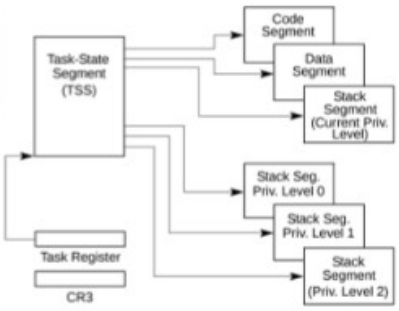
Scheduler: Cambio de nivel de privilegio



```
offset: dd 0
selector: dw 0
...

global _isr32

_isr32:
  pushad

  call pic_finish1
  call sched_nextTask

  str cx
  cmp ax, cx
  je .fin

  mov [selector], ax
  jmp far [offset]

.fin:
  popad
  iret
```

Cada tic del reloj se ejecuta esta rutina de atención de interrupción

Y cada cambio de tarea que haga va a producir un cambio de contexto



Estructura de una tarea